A SOLUTION USING PARALLEL COMPUTING TO CALCULATE PETRI NET MODELS

Bing-lin YANG

Department of Programming Technologies

Reţelele Petri au devenit un instrument popular de modelare şi simulare a fluxurilor de date şi control, în special în sisteme cu activități asincrone şi concurente. Calculul paralel reprezintă o metodă eficientă pentru soluționarea rapidă a problemelor de calcul. În lucrare este descris un algoritm de paralelizare, ce se bazează pe tehnologia MPI, pentru rularea modelelor de Rețele Petri pe clustere de calculatoare.

1. An Introduction to Petri Nets

The basic Petri nets insist of three types of nodes (places, transitions and arcs). Places are used for holding tokens. Transitions are used for controlling token-change between two places. Arcs are used for representing the direction of token flow. Petri net is a type of modeling tools. They were devised for use in the modeling of a specific class of problems, the class of discrete-event systems with concurrent or parallel events. Petri nets model systems, and particularly two aspects of systems, events and conditions, and the relationships among them. In this view, in a system, at any given time, certain conditions will hold. The fact that these conditions hold may cause the occurrence of certain events. The occurrence of these events may change the state of the system, causing some of the previous conditions to cease holding, and causing other conditions to begin to hold. Based on the conditions, Petri net models present events via transitions firing (Tokens move from one place to another though a transition over arcs). If the condition is satisfied, then the transition is validated, which means in next step, the transition can fire. And the number of tokens in two places which are connected to the transition though arcs will be increased or decreased (depends on the direction of arcs which are connected to them). In this way, to simulate or execute a Petri net model, we have to calculate the number of tokens in each place and the status of each transition for each step. All the numbers of each place and status of each transition in a Petri net model is called Marking. The purpose to use Petri net is research the trend of marking-change in order to get the situation of what we are modeling using Petri nets.

2. An Introduction to Parallel Computing

Parallel computing is a form of computation in which many calculations are carried out simultaneously, operating on the principle that large problems can often be divided into smaller ones, which are then solved concurrently. Parallel computing is the major solution to solve problems that need huge calculation. Parallelism has been employed for many years, mainly in high-performance computing, but interest in it has grown lately due to the physical constraints preventing frequency scaling.

Parallel computers can be roughly classified according to the level at which the hardware supports parallelism – with multi-core and multi-processor computers having multiple processing elements within a single machine, while clusters, MPPs, and grids use multiple computers to work on the same task.

The most common four parallel computing structures based on two independent dimensions of instruction and data are: Single Instruction, Single Data (SISD); Single Instruction, Multiple Data (SIMD); Multiple Instruction, Single Data (MISD); Multiple Instruction, Multiple Data (MIMD). The single-instruction-single-data classification is equivalent to an entirely sequential program. The single-instruction-multiple-data classification is analogous to doing the same operation repeatedly over a large data set. This is commonly done in signal processing applications. Multiple-instruction-single-data is a rarely used classification. While computer architectures to deal with this were devised (such as systolic arrays), few applications that fit this class materialized. Multiple-instruction-multiple-data programs are by far the most common type of parallel programs. The part in charge of parallel computing in the project uses multiple-instruction-multiple-data structure and has been implemented with MPI programming library.

3. An Overview of the Project

The project is divided into two parts. One is front-end application which offers researchers graphical user interface software for designing Petri net models, the other is back-end application which is in charge of calculating Petri net models, that is sent from front-end application, based on Petri net execution rules. Figure 1 depicts the two parts.

The project we have been developing is called Visual Membrane Petri Nets system. The front-end application of Visual Membrane Petri Nets system can be run on different operating systems such as Microsoft windows ™, GNU/Linux, and Apple™ Mac OS. The back-end calculation software is responsible for computing the Petri net model in terms of the Petri net execution rules. A rocks computer cluster with 52 processors is being used for calculating in our Petri Nets system, the parallel computing has been implemented using MPI.

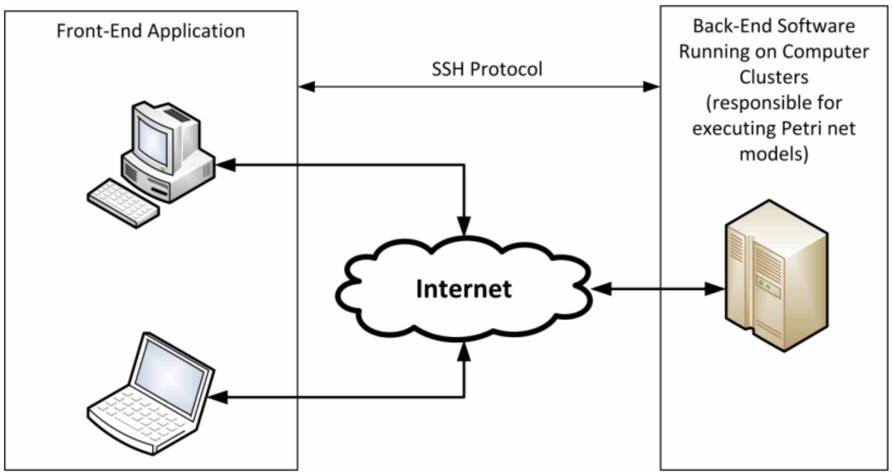
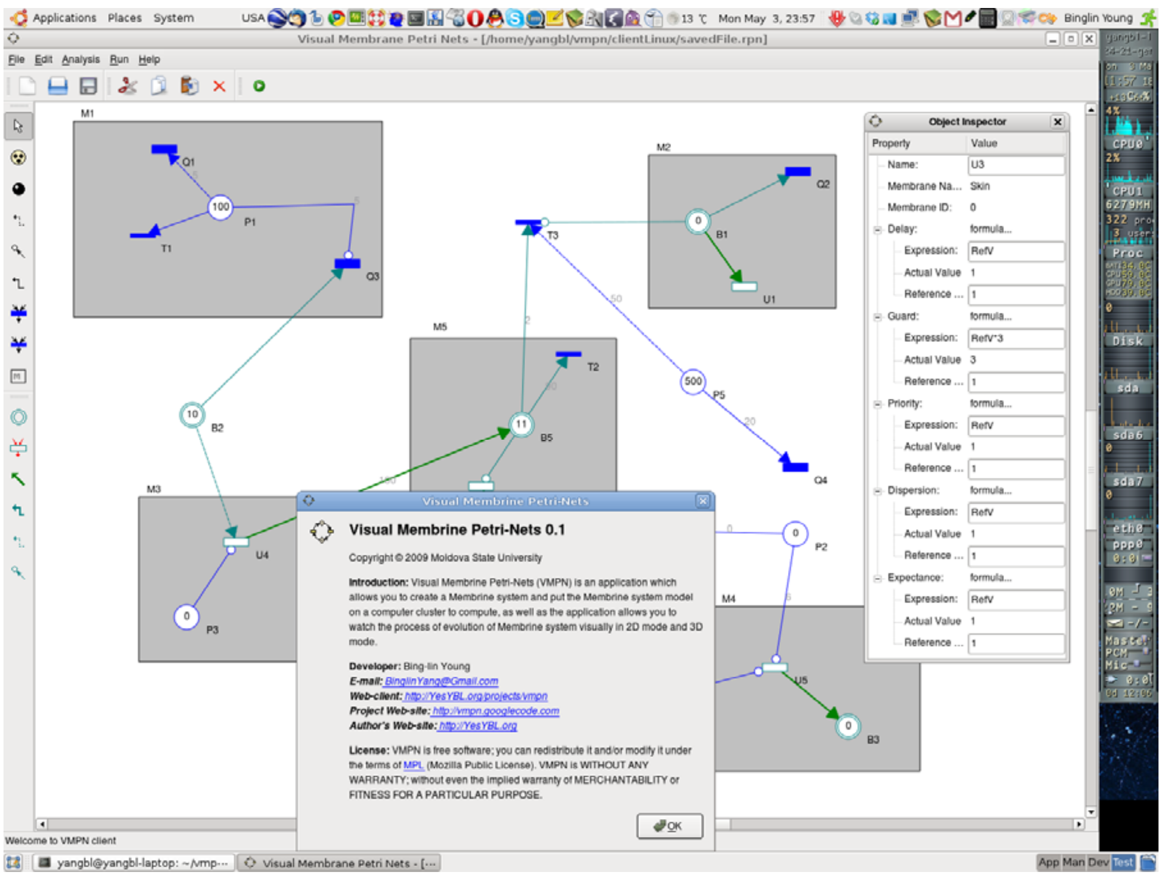

Front-End Application
SSH Protocol
Back-End Software Running on Computer Clusters (responsible for executing Petri net models)
Internet

Figure 1. Two Parts of the Project

4. The Development of Front-end application

We can see the front-end application is a very convenient tool for building Petri net models from figure 2. Researchers can conveniently use a mouse to build a Petri net model on a canvas and set various properties of each node of the model in this software. We enriched the basic Petri nets; therefore, the Visual Membrane Petri Nets can support high level and colored Petri net models. Besides these two points, we introduced an idea of membrane, which could divide a large Petri net model into plenty of subclasses. Based on this dividing, the computation in terms of Petri net execution rules can be divided on various processes that run on different processor as parallel computing.

There are two main types of nodes offered in this tool. One is "discrete nodes" (nodes are in blue color in the figure 2); another one is "continuous nodes" (nodes are in cyan color in the figure 2). Just as their names imply, "discrete nodes" is a set of nodes that only can handle natural numbers, for example, tokens are only represented in discrete natural numbers. Discrete places, Discrete arcs including test arc, inhibitor arc and normal arc, and discrete transitions including timed transition and instant transition, all of them build a discrete system for describing Petri net. "Continuous nodes" is a set of nodes that can handle real numbers; in our system, we use "level" instead of "token" as it can represent a continuous real number. The set of Continuous nodes contains continuous places, continuous transitions, continuous arcs, continuous test arcs and continuous inhibitor arcs. Those nodes build a continuous system for describing Petri nets. One special arc must be pointed out here – flow arc, which is used for connecting continuous place and discrete transitions such as timed transition or instant transition. We can connect discrete part of a Petri net to continuous part of the Petri net through flow arc. But the real number must be converted to a natural number before starting of the movement of tokens or level through this arc.


Applications Places System
Visual Membrane Petri Nets - [/home/yangbl/vmpn/clientLinux/savedFile.rpn]
File Edit Analysis Run Help
M1
Q1
100
P1
T1
Q3
M2
Q2
B1
U1
T3
M5
T2
11
B5
500
P5
Q4
10
B2
M3
U4
0
P3
P2
M4
U5
B3
Welcome to VMPN client
yangbl@yangbl-laptop: ~/vmp···
Visual Membrane Petri Nets - [···
OK

Object Inspector

Property	Value
Name:	U3
Membrane Na...	Skin
Membrane ID:	0
Delay:	formula...
Expression:	RefV
Actual Value	1
Reference ...	1
Guard:	formula...
Expression:	RefV*3
Actual Value	3
Reference ...	1
Priority:	formula...
Expression:	RefV
Actual Value	1
Reference ...	1
Dispersion:	formula...
Expression:	RefV
Actual Value	1
Reference ...	1
Expectance:	formula...
Expression:	RefV
Actual Value	1
Reference ...	1

Visual Membrine Petri-Nets

Visual Membrine Petri-Nets 0.1

Copyright © 2009 Moldova State University

Introduction: Visual Membrine Petri-Nets (VMPN) is an application which allows you to create a Membrine system and put the Membrine system model on a computer cluster to compute, as well as the application allows you to watch the process of evolution of Membrine system visually in 2D mode and 3D mode.

Developer: Bing-lin Young
E-mail: BinglinYang@Gmail.com
Web-client: http://YesYBL.org/projects/vmpn
Project Web-site: http://vmpn.googlecode.com
Author's Web-site: http://YesYBL.org

License: VMPN is free software; you can redistribute it and/or modify it under the terms of MPL (Mozilla Public License). VMPN is WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

Figure 2. The Main Interface of the Front-end Application on Linux Operating System.

4.1. The Execution Rules of the Project

Discrete place:

Function: For holding tokens, this type of place can only hold natural number and its range must be in the interval between the value of its property of "CapacityMin" and the value of its property of "CapacityMax".

Connectible nodes: Test arc, Inhibitor arc and Normal arc.

Continuous place:

Function: For holding the value of level, this kind of place can hold real number and its range must be in the interval between the value of its property of "BoundMin" and the value of its property of "BoundMax".

Connectible nodes: Continuous arc, Continuous Test arc, Continuous Inhibitor arc.

Test arc:

Function: Test arc is used for controlling the status of a transition that is connected to the test arc. Test arc cannot be used for carrying tokens. The status of a transition, which is connected to a test arc, can be enabled if and only if when the value of tokens of a discrete node, which is connected to the test arc, is grater or equals the value of this test arc's weight value. Otherwise the status of the transition must be disabled.

Connectible nodes: Discrete place, Timed transition, Instant transition. (One way connection, place → transition)

Normal arc:

Function: It is used for carrying tokens. Normal arcs can connect from a Discrete place to a Timed transition or Instant transition, and it also can connect to a Discrete place from a Timed transition or Instant transition.

In the case of connecting from a place to a transition, the Normal arc can carry tokens if and only if the value of weight property of the Normal arc is smaller or equals the value of tokens of the place connected to the Normal arc. In each step, the amount of tokens equaling the value of weight should be transported through this Normal arc.

In the case of connecting from a transition to a place, the Normal arc can carry tokens if and only if the status of the transition that connected to the Normal arc is enabled. In each step, the amount of tokens equaling the value of weight should be put into the Discrete place which this Normal arc is connected to on another side.

Connectible nodes: Discrete place, Timed transition, Instant transition. (Bothway connection, place ⟷ transition)

Inhibitor arc:

Function: It is used for controlling the status of a transition, which is connected to the Inhibitor arc, enabled or disabled. The status of a transition can be set enabled if and only if the value of weight property of the Inhibitor is greater than the value of tokens of a place to which the Inhibitor arc connected on another side.

Connectible nodes: Discrete place, Timed transition, Instant transition, Continuous transition. Inhibitor only can start from Discrete place, but can connect to any type of transitions. (One way connection, Discrete place ⟶ any transition)

Continuous arc:

Function: Its function is similar to Normal arc, but it handles real number. Continuous arc may connect from Continuous place to Continuous transition or from Continuous transition to Continuous place.

In the case of a Continuous arc connected from a Continuous place to a Continuous transition, the Continuous arc can be used for transporting if and only if the value of its weigh property is smaller or equals to the value of level property of the Continuous place that is connected to the Continuous arc. In another words, the Continuous transition, which is connected to the Continuous arc, is enabled for next step in this situation.

In the case of a Continuous arc connected from a Continuous transition to Continuous place, the Continuous arc can be used for transporting if and only if the Continuous transition's status is enabled. The value of moving through this Continuous arc equals to the value of its weight property.

Connectible nodes: Continuous place, Continuous transition. (Bothway connection, Continuous place ⟷ Continuous transition)

Flow arc:

Function: Flow arc is used for moving tokens or changing value of level between Continuous place and Timed transition or Instant transition. Actually, this arc builds a channel between Continuous nodes and Discrete nodes. Nevertheless, the type of data must be converted from real number to natural number before moving.

In the case of a Flow arc connected from a Continuous place to a Timed transition or an Instant transition, the Flow arc can carry data if and only if the value of level property of Continuous place is grater or equals to the value of its weight value. In this situation, the status of transition that is connected to this arc is enabled, in next step, the amount of level equaling the value of weight of this Flow arc can be moved.

In the case of a Flow arc connected from a Timed transition or an Instant transition to a Continuous place, the Flow arc can carry data if and only if the status of the transition, which is connected to it, is enabled. The amount of tokens equaling the value of weight of this Flow arc can be put into the Continuous place.

Connectible nodes: Continuous place, Timed transition, Instant transition. (Bothway connection, Continuous place ⟷ Discrete transition)

Continuous Test arc:

Function: Continuous Test arc is used for controlling the status of a Continuous transition which the Test arc is connected to, the arc cannot be used for carrying data. The status of the Continuous transition, which is connected to the arc, can be set enabled if and only if the value of level property of the Continuous place that is connected to the arc is greater or equals to the value of weight property of the Continuous Test arc.

Connectible nodes: Continuous place, Continuous transition. (One way connection, Continuous place ⟶ Continuous transition)

Continuous Inhibitor arc:

Function: Continuous Inhibitor arc is used for controlling the status of a Continuous transition that is connected to the Continuous Inhibitor. This arc cannot be used for carrying data. When the value of level property of the Continuous place that is connected to the arc is lower than the value of weight property of this arc, then the status of the Continuous transition that is connected to this arc can be set enabled. Otherwise, the status should be set disabled.

Connectible nodes: Continuous place, Continuous transition. (One way connection, Continuous place → Continuous transition)

Timed transition:

Function: Timed transition is used for controlling the moving of tokens from one place to another one, or for introducing tokens into a Petri net system or leading out tokens from a Petri net system. Timed transition has "delay" and "guard" two properties. "Delay" represents the rate of speed of tokens' moving. For example, how many steps should be waited for one time transportation. "Guard" property is a boolean number, which holds the status of the Timed transition, enabled or disabled.

Connectible nodes: Normal arc, Test arc, Inhibitor arc and Flow arc.

Instant transition:

Function: The function of Instant transition is similar to Timed transition, but just transporting tokens without waiting. In other words, in the case of when it's enabled, it should transport tokens in each step. The "guard" property of Instant transition is same as Timed transition's.

Connectible nodes: Normal arc, Test arc, Inhibitor arc and Flow arc.

Continuous transition:

Function: Continuous transition is used for transporting value of level (real number), or for leading in data (the value of level) into a Petri net system or leading out data from a Petri net system. Continuous transition contains "delay", "guard" and "expentance" three properties. The functionality of "Delay" and "Guard" is similar to Timed instant. "Expentance" is used for "renewing" the value of weight of arcs that are connected to the Continuous transition.

The new value of weights should be the results of old value of weights multiply the value of expentance, but the new value of weights is only used for calculating the status of the Continuous transition for next step, not kept in the arcs that are connected to the Continuous transition.

Connectible nodes: Continuous arc.

Membrane:

Function: It's used for dividing the current Petri net model or system. Echo portion will be calculated in a unique process on parallel machine. Here, we have to point out that the whole canvas is also counted as a membrane. Thus, the number of membranes should be the number of all membrane nodes on canvas plus one.

4.2. Logical Structure of the Front-end Software

The logical structure of the front-end software is visualized depicted in Figure 3. The front-end software consists of many program modules that are in charge of handling different functions for ensuring researchers building correct Petri net models. A shared or common memory area in which the information related nodes are stored was designed in the application.

The shared memory area is very important in the software, because almost all program modules need to read and write data on this memory area. For example, canvas class has to read the information from this area and create items on canvas in order to let researchers can see them and control them. Property window class (Secondary window class) read data from the shared memory area for displaying properties of echo nodes which users clicked and write new data to this shared memory area after users modified the value of properties in Property window.

Calculator class and Error Checking class have to traverse the data structure in the memory area in order to get the value of properties and formulas, so that the program can renew the value of properties which have to be calculated based on formula expressions that contains other nodes' properties. The directed arrows are used for showing the connection between modules, and the direction represent data flow's orientation (interactions between modules).

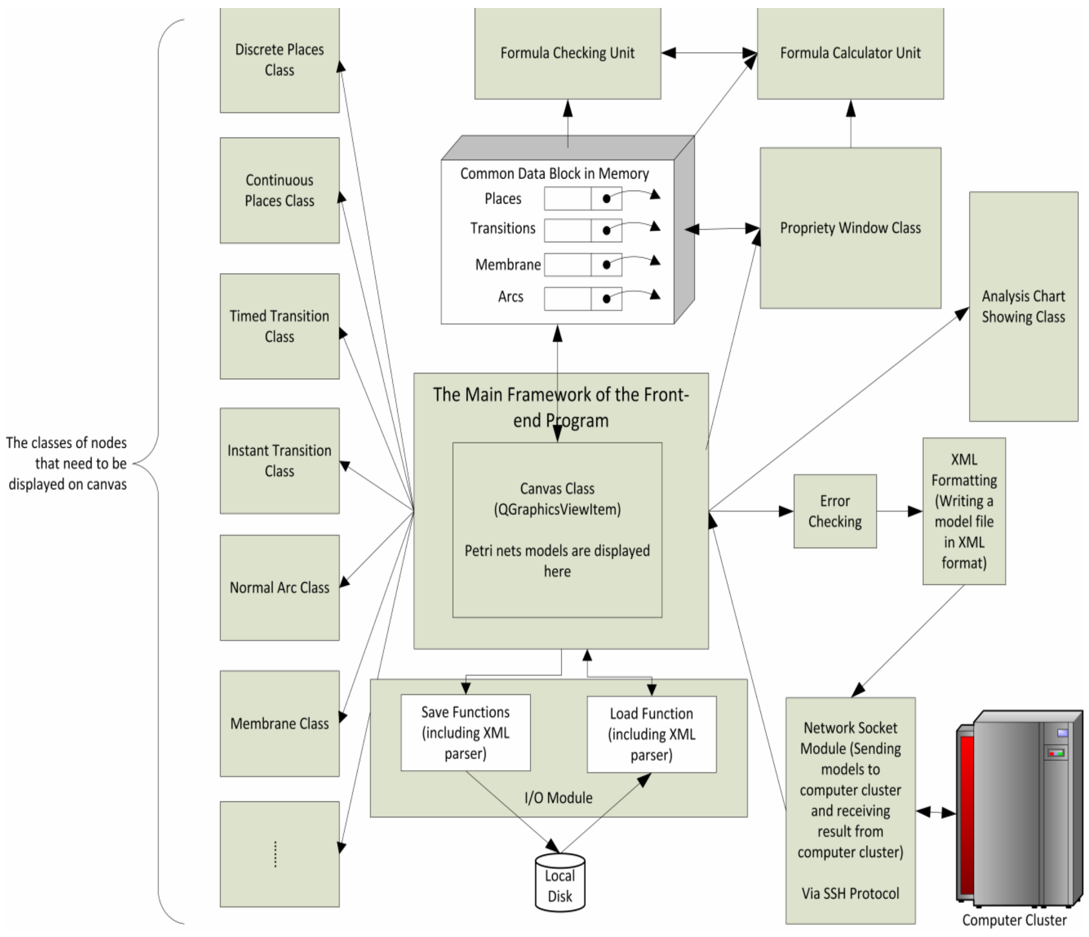

The classes of nodes that need to be displayed on canvas
Discrete Places Class
Continuous Places Class
Timed Transition Class
Instant Transition Class
Normal Arc Class
Membrane Class
Formula Checking Unit
Formula Calculator Unit
Common Data Block in Memory
Places
Transitions
Membrane
Arcs
Propriety Window Class
Analysis Chart Showing Class
The Main Framework of the Front-end Program
Canvas Class (QGraphicsViewItem)
Petri nets models are displayed here
Error Checking
XML Formatting (Writing a model file in XML format)
Save Functions (including XML parser)
Load Function (including XML parser)
I/O Module
Local Disk
Network Socket Module (Sending models to computer cluster and receiving result from computer cluster)
Via SSH Protocol
Computer Cluster

Figure 3. Logical Structure of the Front-end Software.

4.3. File List of the Front-end Program

By reason of the client application is a graphical editor and is designed as a cross-platform software, the current version has been developing using the C++ programming language with Qt4 library. Up to the present moment, the project contains more than 20 source code files (*.cpp, 40,000 lines of code). In this project, almost each C plus plus file is used for implementing a class or a module. Table 1 shows the important source code files' name and purpose. In this list, each header file corresponds to a .cpp file.

Table 1

Name	Purpose
canvas.hh	Implementing a canvas for containing Petri net nodes
chartWindow.hh	Implementing a window for showing analysis chart
common.hh	Containing the common memory area, common algorithmic functions and global variables
formulaCalculator.hh	Calculating formulas
formulaChecker.hh	Checking the error in a formula that user input, including mistyping and logical errors (endless loop formula)
formula.hh	Implementing a GUI window for input a formula
io.hh	saving and loading files
labelItem.hh	Implementing the label item for each nodes in Petri net model
link.hh	Implementing the arcs nodes
mainWindow.hh	Implementing the main interface of the program
membrane.hh	Implementing the membrane nodes
network.hh	Sending and receiving data to and from computer cluster

node.hh	Implementing the places nodes
prefDialog.hh	Implementing preferences dialog window
propEditor.hh	A custom delegate is implemented for handling real number to cooperate with Qt's QTreeWidget class, cause default Qt's delegate item cannot handle real numbers with long fractional part
properties.hh	Implementing a QTreeWidget for handling properties editing needs.
propertiesWin.hh	Implementing a GUI window for handling properties editing needs.
runDialog.hh	Implementing a GUI dialog box for allowing user to calculate Petri net models
simulationDialog.hh	Implementing a GUI dialog box for getting result back from computer cluster and simulate it
trans.hh	Implementing the transitions nodes.

5. Design of the Back-end Program

The Back-end program is in charge of calculating Petri net models in parallel. The program is designed running on a Rock Computer Cluster with 52 processors. Because the features of Petri nets and computer cluster, we designed this parallel program in Multiple-Instruction-Multiple-Data mode on distributed memory architecture.

Domain of data (Petri net models) is divided by membranes. How many tasks will be involved depending on how many membranes in the current model? In other words, each task that runs on a processor will only handle with transitions and places which are located in a membrane and the membrane's id is equal to the number of process ID. In this way, the data is decomposed. The procedure of calculating Petri nets is flow-dependent and anti-dependent, which means that in each step, when one processor finishes its own calculation, the process on that processor must exchange data with other processes on other processors.

5.1. Data That Need to Be Exchanged among Processes

All formulas are not changed in the whole procedure of calculation, thus, all information related to formulas does not need to be exchanged. From this point, we can get that all data related to arcs do not need to be exchanged, because nodes of arcs only contains formulas. The status of each transition is the critical data in exchanging, because the current transitions' statuses control the following calculation of place node. The number of tokens/levels in each place is changed in each step, and value of the numbers also affects the statuses of transitions in next step; therefore, Petri nets are flow-dependent and anti-dependent. Because of the reasons above, we have to exchange the statuses of transitions in the beginning of each step, and then calculate place nodes, after the calculation, exchange data of place nodes, and then calculate the statuses of all transition nodes for next step. In this way, calculate Petri net models step by step until the number of step reaches the maximum number or the marking no changes any more.

5.2. Concrete Calculation Method

The Initial step (Step 0):

On the computer cluster, the master process (rank = 0) read the data file (which is in XML format, and was sent from the front-end program) from hard-disk and parse it, build data-structure in memory, then send the whole Petri net model to other processes. After that, all processes got the Petri net model and ready to calculate the model.

In the beginning of executing the model, each process has to calculate the statues of transitions that are in the membrane with the number of its ID equaling to the process's ID. (Probable, the data, such as the value of tokens and weight property, of arcs and places that are located in other membranes will be used in the procedure of this calculation. Thus, the whole Petri net model must be completely stored in each process's data area.) After the calculation, the new status of transitions must be set (saved). Then starts exchanging the status of transitions, each process has to send its transitions' new statues to other processes and receive all new statues of transitions from all processes, so that all processes can get a new status of the whole Petri net model.

Right now, master process has to save the current marking in memory and back up the marking as a copy in memory for comparing marking in next step.

The first step (Step 1):

Based on the new statues of transitions in the whole Petri net model, each process can start to calculate the places in its own membrane. After calculating places, the number of token or level will be updated, and then those new values have to be exchanged. Using MPI_Bcast function call exchange the data in order to make that all processes can get new values of places. Then, the statues of transitions should be calculated for next step just like in the initial step, so that the new marking of the whole Petri net model can be stored in all process. Master process has to record the marking in memory as the execution result. After that, each process has to compare the value of new marking with the value of previous marking in order to check if the execution over or not. If the value of new marking is not same as previous one, which means the calculation should be continue, the calculation must get into second step.

The second step and step N:

Repeat the same operation as in the first step, and the rest can be done in the same manner. When the calculation finishes, the master process has to write down the execution result into a XML file on hard-disk, so as to the front-end program can retrieve the result via SSH copying. Flow chart of this procedure is figure 4.

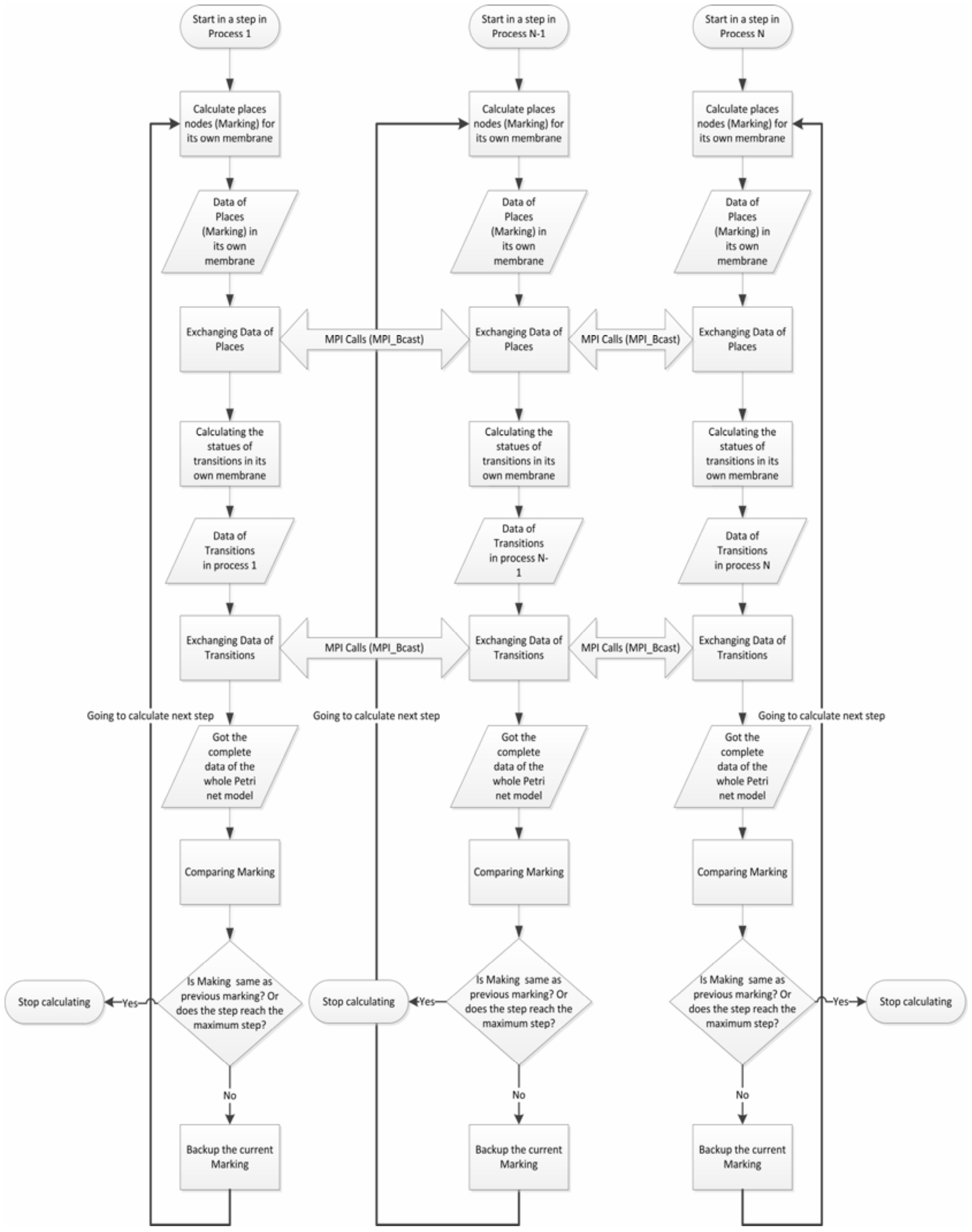

Start in a step in Process 1
Start in a step in Process N-1
Start in a step in Process N
Calculate places nodes (Marking) for its own membrane
Data of Places (Marking) in its own membrane
Exchanging Data of Places
MPI Calls (MPI_Bcast)
Calculating the statues of transitions in its own membrane
Data of Transitions in process 1
Data of Transitions in process N-1
Data of Transitions in process N
Exchanging Data of Transitions
Going to calculate next step
Got the complete data of the whole Petri net model
Comparing Marking
Is Making same as previous marking? Or does the step reach the maximum step?
Yes
Stop calculating
No
Backup the current Marking

Figure 4. The Flow Chart of Parallel Algorithm

5.3. Return the Result of Execution

The front-end program can copy the result file from computer cluster to researchers' local computer, then the "analysis chart showing module" will parse the XML file, get the data of marking for each step, then give the final analyzed result to researchers. Figure 5 shows a simple Petri nets model's result in analysis chart.

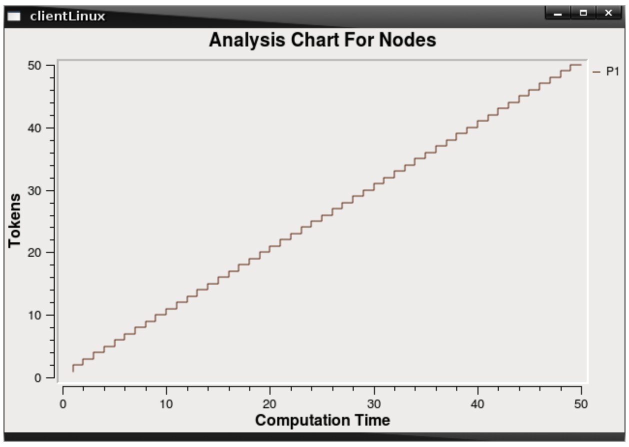


Figure 5. An Example of Analyzed Chart of Marking Changes.

Bibliography:

1. James L. Peterson. Petri Nets. Computing Surveys, vol. 9, No.3, September 1977, USA.
2. Profir A, Gutuleac E., Ciurganov D., Opinca C., Danu S. Modeling of molecular genetic mechanisms involved in the development of cardiovascular diseases using Visual Membrane Petri Nets. "Medical-Surgical Journal" of Iasi, "Gr.T. Popa" University of Medicine and Pharmacy, Iasi, Romania, November 2007, p.157-160.
3. Jasmin Blanchette, Mark Summerfield. C++ GUI Programming with Qt 4. June 21, 2006.
4. Cheng G. L., An H, Cheng L, Zheng Q. L., Dan J. L. The Practice of Parallel Computing Algorithm. Higher Education Press, Jan 2004, p.58-97, p.107-127, China.
5. Trolltech, Qt Reference Documentation (Open Source Edition).

Prezentat la 23.02.2011